IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application for:

PARTITIONED VECTOR PROCESSING

Inventors: James M. Norris
Philip E. May
Kent Donald Moat
Raymond B. Essick IV
Brian Geoffrey Lucas

Docket Number: CML00104D

PARTITIONED VECTOR PROCESSING

PRIORITY CLAIM

This application is a continuation-in-part application that claims priority under 35 U.S.C. 120 to co-pending U.S. patent application serial number 10/184,583 titled "Reconfigurable Streaming Vector Processor", filed June 28, 2002, Art Unit 2183, Examiner Charles A. Harkness, being further identified by Attorney Docket No. CML00107D, which is herein incorporated by reference.

CROSS REFERENCE TO RELATED APPLICATIONS

This application is related to patent application Attorney Docket No. SC13071TH titled "Data Processing System Using Multiple Addressing Modes for SIMD Operations and Method Thereof" filed on the same date as this application, which is assigned to the current assignee hererof.

FIELD OF THE INVENTION

This invention relates generally to the field of vector processing. More particularly, this invention relates to a method and apparatus for accessing partitioned memory for vector processing.

BACKGROUND OF THE INVENTION

Many new applications being planned for mobile devices (multimedia, graphics, image compression/decompression, etc.) involve a high percentage of vector

computations. One limitation on the computation rate of these applications is the speed of accessing vector or matrix data stored in memory.

One approach to accessing vector data is to specify the starting address in memory of the data, the size of each data element (in bits) and the separation between consecutive data elements (the "stride"). This approach allows sequential data to be accessed, but cannot be used where the elements are not separated by a constant amount. So, for example, the approach cannot be used if parts of a data vector are stored in different memory partitions. For example, a two-dimensional image may be stored in consecutive memory locations, one row at a time. The memory addresses of a data vector representing a sub-block are not separated by an equal amount.

A further approach, which has application to the processing of sparse data matrices, is to generate vectors specifying the locations of the non-zero matrix elements in memory. While this method provides the flexibility required for specialized Finite Element calculations, it is more complex than required for most multimedia applications on portable devices.

A still further approach uses L1 and L2 memory caches to speed memory access. The data is pre-fetched in blocks defining the starting address, block size, block count, stride and stride modifier. The stride modifier allows diagonal elements of a data matrix to be accessed. However, the approach cannot be used unless the data elements are separated by a constant amount. Further, the approach does not allow for data access to start part way through a block without modifying the block structure.

SUMMARY

The present invention relates generally to a method and apparatus for accessing a set of vector elements in a partitioned memory. Objects and features of the invention will become apparent to those of ordinary skill in the art upon consideration of the following detailed description of the invention.

In accordance with one aspect of the invention, an address calculator is provided for calculating memory addresses in a partitioned memory in a processing system having a processing unit, input and output units, a program sequencer and an external interface. The address calculator includes a set of storage elements and an arithmetic unit for calculating a memory address of a vector element dependent upon the values stored in the storage elements and the address of a previous vector element. The storage elements store STRIDE, SKIP and SPAN values and, optionally, a TYPE value, relating to the spacing between elements in the same partition, the spacing between elements in the consecutive partitions, the number of elements in a partition and the size of a vector element, respectively. In accordance with an embodiment of a method of the invention, an element address, a first counter indicative of the number of elements of the vector elements in the first memory and a second counter indicative of the number of elements in the vector elements are initialized. Then, while the second counter indicates that not all of the vector elements have been accessed, the memory is accessed at the element address and the second counter is stepped. If the first counter indicates that at least one vector element remains in the partition, the element address is incremented by an amount related to the STRIDE or the product of the TYPE and STRIDE values and the first counter is stepped. Otherwise, the

element address in incremented by an amount related to the SKIP or the product of the TYPE and SKIP values and the first counter is reset dependent upon the SPAN value, which indicates the number of elements of the vector elements in a partition.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as the preferred mode of use, and further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawing(s), wherein:

FIG. 1 is a representation of a processing system in accordance with an embodiment of the present invention.

FIG. 2 is a representation of an addressing system in accordance with an embodiment of the present invention.

FIG. 3 is a representation of a partitioned memory in accordance with an embodiment of the present invention.

FIG. 4 is a representation of a partitioned memory in accordance with a further embodiment of the present invention.

DETAILED DESCRIPTION

While this invention is susceptible of embodiment in many different forms, there is shown in the drawings and will herein be described in detail one or more specific embodiments, with the understanding that the present disclosure is to be considered as exemplary of the principles of the invention and not intended to limit

the invention to the specific embodiments shown and described. In the description below, like reference numerals are used to describe the same, similar or corresponding parts in the several Views of the drawings.

Vector processing may be performed by general-purpose processor or specialized processor. An example is the Reconfigurable Streaming Vector Processor (RVSP) described in the co-pending patent application serial number 10/184,583 titled "Reconfigurable Streaming Vector Processor", filed June 28, 2002, which is hereby incorporated herein by reference.

An exemplary processing unit incorporating an addressing system of the present invention is shown in **FIG. 1**. Referring to **FIG. 1**, the system includes a processing unit 10, which may comprise a number of functional elements and storage for intermediate results, an input unit 12 and an output unit 14. The input and output units incorporate addressing hardware or arithmetic unit 100 that will be described in more detail below with reference to **FIG. 2**. The function of the input unit 12 is to retrieve data elements via an external interface 16 (e.g. a system bus) and pass them to the processing unit 10. The function of the output unit 14 is to receive data elements from the processing unit 10 and pass them to the external interface 16. The system also includes a program sequencer 18 that controls the operation of the processing unit via link 20. The program sequencer 18 also controls the input and output units via links 22 and 24 respectively. The program sequencer executes a program of instructions that may be stored locally in a memory. The program of instructions may be received via the external interface 16, or via a separate interface. In the latter case, the processing system may have both a memory interface and a host interface.

An important element of a processor is its ability to access a vector of data elements stored in memory. Memory access is simplified when data elements are stored sequentially in memory. The data may be interleaved, in which case consecutive elements are not contiguous but are separated by an amount called a STRIDE. The STRIDE may be measured in a variety of different units, such as the number of elements between elements to be accessed, the number of words, the number of bytes or the number of bits. The STRIDE may be a fractional number to enable to access of subwords, for example. When large data structures are involved, data may be stored in different memory partitions. Also, when two- or three-dimensional data structures are stored in a linear memory, each row or column of the structure may be considered to be stored in a separate partition. Consecutive elements stored in different partitions may be separated by an amount that is different from the stride. This amount will be referred to as the "skip". Prior techniques do not use a "skip" value and so cannot be used where the elements are not separated by a constant amount, as when parts of a data vector are stored in different memory partitions. Prior techniques require the issuance of one or more additional instructions to access multiple memory partitions. This results in reduced performance and more complicated programming.

When accessing a sub-array from 2-dimensional array, the skip value may be used to move an address pointer to a new row or column of the array. When accessing a sub-array from 3-dimensional array, a second skip value may be used to move an address pointer to a new level of the array.

An exemplary embodiment of the address calculator 100 of present invention is shown in **FIG. 2**. Referring to **FIG. 2**, the address calculator 100 comprises a set of storage elements 102. The storage elements will be referred to as registers in the sequel, but may be other types of memory circuits or devices. The storage elements 102 include a TYPE register 104, a STRIDE register 106, a SKIP register 108 and a SPAN register 110.The registers are accessed by an arithmetic unit 112. The arithmetic unit may, for example, comprise a state machine and adder. The arithmetic unit 112 is initialized by a set of initialization values 114 that include the start address, denoted by EA_START, of a vector of data to be accessed, the initial value, denoted by LEFT_START, of a counter that indicates the number of data elements remaining in the first partition, and the total number of data elements, denoted by TOTAL, to be accessed in the memory. Once initialized, the arithmetic unit 112 is operable to calculate the address of a current data element in memory from the address of the previous element. The current address is stored in address pointer 116 and may be output at 118 to access the memory. The address calculator 100 may be used in concert with a pre-fetch architecture, such as a cache, so as to mitigate the adverse effects of slower memory. In this way, a processor may access data in almost every clock cycle, and be used with cheaper (slower) memory in cost sensitive applications.

The register values TYPE, STRIDE, SKIP and SPAN may be controlled by instructions sent from a program sequencer. The initial values EA_START, LEFT_START and TOTAL may be set in a similar fashion. If any of the values TYPE, STRIDE, SKIP, SPAN or LEFT_START is not specified, default values may

be used. For example, the default values may assume that the data is stored in memory in a single partition of contiguous data.

A diagrammatic representation of an exemplary partitioned memory is shown in **FIG. 3**. In this simplified example, the memory has three partitions (PARTITION 0, PARTITION 1 and PARTITION 2). The data vector to be accessed is interleaved so that, within each partition, every third memory element is an element of the vector. The address of the first data element is indicated as EA_START. Five data elements are stored in each memory partition, so the LEFT counter is initialized to 5. The total number of elements to be accessed is TOTAL=15, so a second counter is initialized with the value 15 and is decremented as each vector element is accessed. After the first element is accessed, the LEFT counter is decremented to 4, indicating that only 4 values remain in the current partition, and the TOTAL counter is decremented to 14. It will be apparent to those skilled in the art that vector elements may be counted by incrementing or decrementing counters. The address of the next element is calculated by adding the product of the STRIDE value and the TYPE value to the address of the current element. In this example, STRIDE=3, since every third element is to be accessed. TYPE denotes the length (in bits for example) of each data value. The process continues until the last element of the partition is accessed. The LEFT value is than decremented from 1 to 0. When the LEFT value goes to zero, the next memory address is calculated by adding the product of the SKIP value and the TYPE value to the current address. In this example, SKIP=5. The address then points to the first value in PARTITION 1. The LEFT value is reset to 5, to indicate that 5 values

remain in PARTITION 1. This process continues until all vector elements (15 in this example) have been accessed.

A further example of a partitioned memory is shown in **FIG. 4**. Referring to **FIG. 4**, the same partitioned data structure is used, but in this example the starting address EA_START is part way through a partition, rather than at the start of a partition. The arithmetic unit is initialized with LEFT=4 and TOTAL = 14. All of the other components of the partitioned memory remain as in the previous example. Since the data structure is preserved, this approach allows access to any vector element at anytime while still maintaining access to other elements.

A pseudo-code listing of an embodiment of the arithmetic unit (112 in **FIG. 2**) is given below.

```
// Initialization
EA = EA_START              //start address
LEFT = LEFT_START          //position in partition
COUNT = TOTAL              //element counter

// Loop over all data elements
WHILE COUNT > 0
     COUNT = COUNT - 1     //decrement element counter
     EA = EA + STRIDE * TYPE  // new address
     IF LEFT > 0           //in same partition
          LEFT = LEFT - 1
     ELSE                  //move to next partition
          EA = EA + SKIP * TYPE    //new address
          LEFT = SPAN      //reset partition position
END WHILE
```

If the STRIDE and SKIP values specify memory values, rather than a number of elements, the TYPE value is unity and may be omitted. In the embodiment described in the pseudo code above, the STRIDE value is applied after each element is addressed. In a further embodiment, the STRIDE value is not applied at the end of block, and the SKIP value modified accordingly. For example, for uniformly spaced elements, SKIP=0 for the first embodiment, while SKIP=STRIDE for the second embodiment. The second embodiment may be described by the pseudo code given below.

```
// Initialization
EA = EA_START                  //start address
LEFT = LEFT_START              //position in partition
COUNT = TOTAL                  //element counter

// Loop over all data elements
WHILE COUNT > 0
     COUNT = COUNT - 1         //decrement element counter
     IF LEFT > 0               //in same partition
          EA = EA + STRIDE * TYPE  // new address
          LEFT = LEFT - 1
     ELSE                      //move to next partition
          EA = EA + SKIP * TYPE    //new address
          LEFT = SPAN          //reset partition position
END WHILE
```

In the special case, where an equal number of elements are to be accessed from each partition, the LEFT value is initialized with SPAN value, where SPAN is

the number of elements in a partition. Equivalently, the number of elements accessed in a partition may be counted and compared with the value SPAN, to determine if a skip should be made to the next partition.

In a further embodiment of the invention, the SKIP and STRIDE values denote the number of bits between elements, rather than the number of elements (words of length TYPE). In this embodiment, the TYPE parameter is not required.

Data from a three-dimensional structure (such as a video clip) is partitioned in two levels. The first level represents to rows of a particular image while the second level represents the image at a different time. A pseudo-code listing of a further embodiment of the arithmetic unit (112 in **FIG. 2**) for accessing three-dimensional data is given below.

```
// Initialization
EA = EA_START              //start address
LEFT = LEFT_START          //position in partition 1
LEFT2 = LEFT2_START        //position in partition 2
COUNT = TOTAL              //element counter

// Loop over all data elements
WHILE COUNT > 0
     COUNT = COUNT - 1     //decrement element counter
     EA = EA + STRIDE * TYPE  // new address
     IF LEFT > 0           //in same level 1 partition
          LEFT = LEFT - 1
     ELSE                  //move to next partition
          EA = EA + SKIP * TYPE    //new address
          LEFT = SPAN      //reset partition position
```

```
                IF LEFT2 > 0    //in same level 2 partition
                     LEFT2 = LEFT2 - 1
                ELSE            //move to next partition
                     EA = EA + SKIP2 * TYPE    //new addr.
                     LEFT2 = SPAN2 //reset position
                ENDIF
           ENDIF
      END WHILE
```

In this embodiment an additional counter LEFT2 and additional parameters SPAN2 and SKIP2 are required to allow for the extra dimensional. It will be clear to those of ordinary skill in the art how the technique may be expanded to access higher dimensioned data structures.

Those of ordinary skill in the art will recognize that the present invention has application in general purpose processors as well as microprocessor based computers, digital signal processors, microcontrollers, dedicated processors, and other hardware accelerators including vector processors.

The present invention, as described in embodiments herein, is implemented using hardware elements operating as broadly described in pseudo-code form above. However, those skilled in the art will appreciate that the processes described above can be implemented in any number of variations. For example, the order of certain operations carried out can often be varied, additional operations can be added or operations can be deleted without departing from the invention. Such variations are contemplated and considered equivalent. Further, the invention may be constructed using custom circuits, ASIC's and/or dedicated hard-wired logic or alternative equivalents.

While the invention has been described in conjunction with specific embodiments, it is evident that many alternatives, modifications, permutations and variations will become apparent to those of ordinary skill in the art in light of the foregoing description. Accordingly, it is intended that the present invention embrace all such alternatives, modifications and variations as fall within the scope of the appended claims.

What is claimed is: